DCAngel

A comprehensive workload characterization

Tool

USER'S MANUAL

October 7th 2011

Revision Sheet

Release No.	Date	Revision Description
Rev. 1.0	07/10/2011	DCAngel v1.0 first packaging

USER'S MANUAL

TABLE OF CONTENTS

CONTENTS

1. Introduction 1

1.1. Targeted Audience 1

1.2. Structure of the document 1

1.3. Further Readings 1

2. DCAngel 2

2.1. Quick introduction 2

2.2. Available implementations 2

3. Getting started 3

3.1. Overview 3

3.2. Prerequisites 3

3.2.1. Linux Kernel Version 3

3.2.2. perf 3

3.2.3. Python 4

3.2.4. Numpy 4

3.2.5. Matplotlib 4

3.2.6. CPU 4

3.2.7. SSH 4

3.2.8. Setup passphraseless ssh 4

3.2.9. Network 5

3.3. Deploying DCAngel 5

3.3.1. Configuration 6

3.4. Running DCAngel 6

3.4.1. Add your own application 6

3.4.2. Start DCAngel....6
3.4.3. Get result....7

4. Building your own DCAngel....10

4.1. CPU....10
4.2. Separate DCAngel....11

Appendix A –Metrics collected by DCAngel....12

Appendix B – DCAngel database table structure....14

1. Introduction

DCAngel is a comprehensive workload characterization tool, It can collect performance metrics and then write them into database for further analysis and visualization.

DCAngel v1.0 package brings some simplicity in terms of installation, deployment and monitoring.

1.1. Targeted Audience

This document is targeting two types of audiences:

- People who just want to use *DCAngel* to collect performance metrics. This is for those who will directly use the provided *DCAngel* directly to deploy it on their cluster.
- People who would like to modify the sources to fit their particular needs.

1.2. Structure of the document

This document goes on the following route:

- A detailed introduction will be given in Section 2, for people who have never used *DCAngel* before.
- How to install *DCAngel v 1.0* is introduced in Section 3, for people who are not going to make any change to the provided *DCAngel*.
- How to build an appliance on your own needs can be found in Section 4

1.3. Further Readings

The following links give more in-depth details about technologies used in *DCAngel v1.0*.

- **Perf** : https://perf.wiki.kernel.org/index.php/Main_Page
- **Sqlite3:** http://www.sqlite.org/
- **Numpy:** http://numpy.scipy.org/
- **Matplotlib:** http://matplotlib.sourceforge.net/

2. DCAngel

2.1. Quick introduction

DCAngel is a comprehensive workload characterization tool. It can collect performance metrics and then write them into database for further analysis and visualization. We use *perf* to collect performance counters' data and sqlite3 as the database. The whole tool is developed in python.

2.2. Available implementations

You may find available information and descriptions about older *DCAngel* versions at its home page (**http://prof.ncic.ac.cn/DCBenchmarks**). If newer version implemented, it will be appended.

If you find some bugs, please contact us via jiazhen@ncic.ac.cn.

If you successfully implement it on your own platform, please let us know.

If you have some novel idea, you might share with us.

3. Getting started

In this part, you will drive right into the configuration and running part, supposing you don't want to modify the provided *DCAngel*.

3.1. Overview

DCAngel adopts a simple master—slaves model as shown in Figure 1.

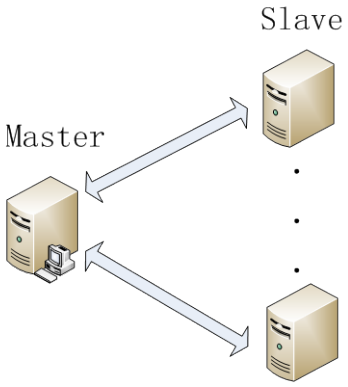

Figure 1 Architecture of DCAngel

- **Master**: starting your application and controlling Slaves to collect metric data.
- **Slave**: collecting metric data and sending those data to Master.

3.2. Prerequisites

The provided *DCAngel v1.0* relies on *perf* and *Python*. In this part, we focus on how you can use what is provided in the *DCAnge-v1.0* package, for deeper information you may refer to the Building part in section 4.

3.2.1. Linux Kernel Version

For this step, you need to get the root privileges for your Linux systems.

We need to build a linux kernel whose version is 2.6.31 or newer for all the **Slave** nodes, because those kernels support *perf_events* port, which is used by *perf*. Make sure that *perf_events* options are selected in the configuration before compiling the kernel.

3.2.2. perf

For *perf* , users should get a linux kernel source code whose version is 2.6.31 or newer on all **Slave** nodes and then enter the directory *tools/perf*. After that, users

should execute the following commands to install *perf*:

```
make
make install
```

3.2.3. Python

All the linux systems need *Python* whose version is 2.7. Older or newer versions haven't been verified in our system.

3.2.4. Numpy

The **Master** node needs *Numpy* (http://numpy.scipy.org/), which is the fundamental package needed for scientific computing with Python. You may need the following libraries or tools before installing *Numpy*:

atlas, python-nose, lapack, blas, libgfortran, python-dateutil, python-matplotlib, python-tz, python-setuptools

3.2.5. Matplotlib

The **Master** node needs *matplotlib*(http://matplotlib.sourceforge.net/), which is a python 2D plotting library.

3.2.6. CPU

For this version, the **Slave** nodes' CPU type must be as below:

1. Intel Xeon processor 3000, 3200, 5100, 5300 series
2. Intel Core 2 duo processor

If you use other CPUs, you may go over the CPU part in section 4.

3.2.7. SSH

SSH must be installed and *sshd* must be running. To run the *DCAngel* scripts that manage remote daemons, please make sure that you can *ssh on* remote nodes without a passphrase

3.2.8. Setup passphraseless ssh

Master node must *ssh* to **Slave** nodes without a passphrase, Now check that.

```
$ ssh localhost
```

If you cannot ssh to nodes without a passphrase, execute the following commands at Client node:

```
$ ssh-keygen -t dsa -f $HOME/.ssh/id_dsa -P ''
This should result in two files, $HOME/.ssh/id_dsa (private key) and
```

$HOME/.ssh/id_dsa.pub (public key).
Copy $HOME/.ssh/id_dsa.pub to **Slave** nodes
On those nodes run the following commands:
$ cat id_dsa.pub >> $HOME/.ssh/authorized_keys2
$ chmod 0600 $HOME/.ssh/authorized_keys2
Depending on the version of OpenSSH the following commands may also be required:
$ cat id_dsa.pub >> $HOME/.ssh/authorized_keys
$ chmod 0600 $HOME/.ssh/authorized_keys
An alternative is to create a link from authorized_keys2 to authorized_keys:
$ cd $HOME/.ssh && ln -s authorized_keys2 authorized_keys
On the **Master** node test the results by ssh'ing to other nodes:
$ ssh -i $HOME/.ssh/id_dsa server

This allows ssh to access to the nodes without specifying the path of the id_dsa file as an argument to ssh each time.

3.2.9. Network

This should come as no surprise, but for the sake of completeness we have to point out that all the machines must be able to reach each other over the network. The easiest is to put all machines in the same network with regards to hardware and software configuration, for example, you may connect machines via a single hub or switch and configure the network interfaces to use a common network such as 192.168.0.x/24.

To make it simple, we will access machines using their hostname, so you should write the IP addresses and corresponding hostnames into /etc/hosts. The following is an example.

```
#/etc/hosts
10.10.104.47 gd47
10.10.104.48 gd48
10.10.104.49 gd49
10.10.104.50 gd50
```

3.3. Deploying DCAngel

You're suggested creating a new user for all Linux systems, and log in with the new user to do the following step. To make it simple, we just assume the new user you created for the tool is **ans42** with the password 'a'.

The user should download the *DCAngel-v1.0* package to the **Master** node using the user *ans42*. We assume that you put the decompressed package in the directory of *$DCAngel*. All the following operations should be done in **Master** node.

3.3.1. Configuration

To deploy *DCAngel*, you should first configure the *$DCAngel/common.mk* file as follow.

```
uname = ans42      # the user's name for the tool
upwd = a           # the corresponding password of the user
node = gd48,gd49,gd88    # the hostname of Slave nodes
```

Do not change other configurations in this file.

At last, execute "**make deploy**" and "**source ~/.bashrc**". Then *DCAngel* will be deployed on all nodes. The deployment time depends on the number of nodes and the machine's hardware configuration. It maybe needs a few minutes.

3.4. Running DCAngel

3.4.1. Add your own application

Enter the *$DCAngel/exp* directory and edit the Makefile file.

```
54 ################start your application here ##############################
55 %/application:
56              @echo 'application start'
57              sleep 5
58
59 ########################################################################
```

You should add your application command from line 56. The application here is *sleep*, and the *DCAngel* will collect **Slave** nodes' metric data when Master node sleep for 5 seconds.

3.4.2. Start DCAngel

Enter the *$DCAngel/exp* directory and edit the run-test.sh file.

```
11 #--------write test here-------------------------#
12 report #(app).#(annotation).#(workload)
```

The string of "#(app).#(annotation).#(workload)" is the directory's name for your application's performance data. You should replace #(app), #(annotation), #(workload) with your own words. But you must make your command in the same format as above (Note that we use a dot(.) to connect two sections). #(app) means your applications name; #(annotation) is the annotation of this test; #(workload) means the information or characterization of this application's workload. Those strings are just used to

distinguish different test, and users can use any words they like. For the sleep application, you can write following statement to run-test.sh file:

```
report sleep.5seconds.NULL
```

"*sleep*" means that your application's name is "sleep"; the annotation "*5seconds*" means it sleeping for 5 seconds and "*NULL*" means this workload dose nothing in this 5seconds.

After that, you should run the following command under *$DCAngel/exp* directory to start the test and let *DCAngel* collect performance data.

```
$ make test
```

3.4.3. Get result

After your application running over, you can use *DCAngel* to get performance date, aggregate data and visualize date.

Figure.2 shows the high-level diagram of *DCAngel*. It stores performance data in a relational database managed by SQLite3 that supports the extended SQL statements. Users can access those data through the extended SQL statements.

All the tests' log and performance data collected by *DCAngel* can be find in *$DCAngel/exp/log/#(app).#(annotation).#(workload)* directory, where the *#(app).#(annotation).#(workload)* is part of the make command you input to start a test. For example, if you use the make command in section 3.4.2, the log can be found at *$DCAngel/exp/log/sleep.5second.NULL*. In that directory, there will be a file named *exp-report* if the test have finished. The file is always empty, as it is just a signal to tell users whether the whole test has finished. The exp-log file records the start time and end time of the test. The *hmon* directory collects performance data of **Slave** nodes. We have already included an example test result in the *DCAngel* package, which you can find at *$DCAngel/exp/log/ sleep.GivenExample.NULL*.

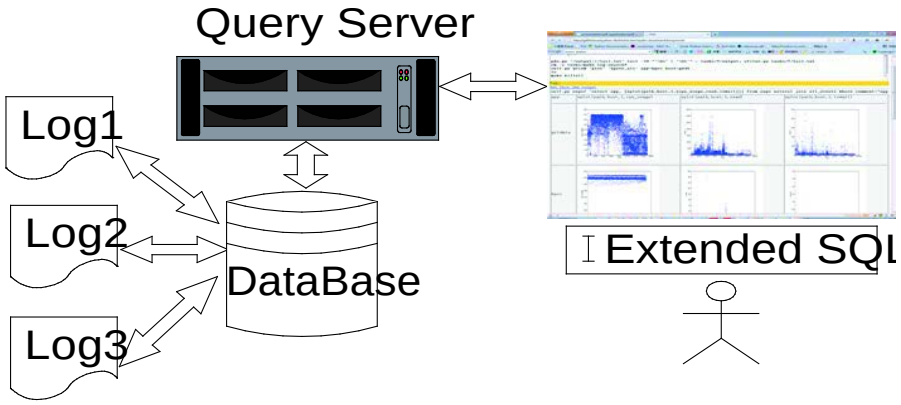

Figure 2 High Level Diagram of DCAngel

We can get data through a browser using *DCAngel*. For this version, the only browser we supported is *FireFox*. First, we should start the service by executing the following commands.

```
enter the directory $DCAngel/python-lib/fsh/:
$ cd $DCAngel/python-lib/fsh
```

Start the service: ./psh.py *#port*. For the *#port*, we use 8002 as an example.
$./psh.py 8002

And then we can visit *DCAngel*'s browser port through the address (do not forget the slash after "fsh"):

http://Client node ip address :port/$DCAngel/exp/cmds/fsh/

The *$DCAngel* above is the location of DCAngel-v1.0 package.

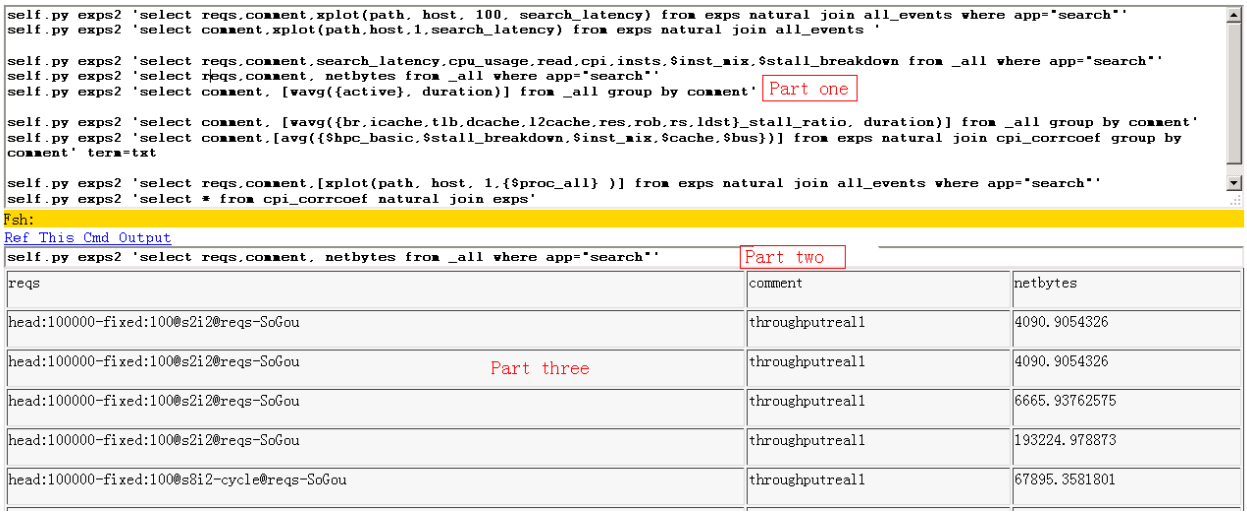
self.py exps2 'select reqs,comment,xplot(path, host, 100, search_latency) from exps natural join all_events where app="search"'
self.py exps2 'select comment,xplot(path,host,1,search_latency) from exps natural join all_events '
self.py exps2 'select reqs,comment,search_latency,cpu_usage,read,cpi,insts,$inst_mix,$stall_breakdown from _all where app="search"'
self.py exps2 'select reqs,comment, netbytes from _all where app="search"'
self.py exps2 'select comment, [wavg({active}, duration)] from _all group by comment'
Part one
self.py exps2 'select comment, [wavg({br,icache,tlb,dcache,l2cache,res,rob,rs,ldst}_stall_ratio, duration)] from _all group by comment'
self.py exps2 'select comment,[avg({$hpc_basic,$stall_breakdown,$inst_mix,$cache,$bus})] from exps natural join cpi_corrcoef group by comment' term=txt
self.py exps2 'select reqs,comment,[xplot(path, host, 1,{$proc_all})] from exps natural join all_events where app="search"'
self.py exps2 'select * from cpi_corrcoef natural join exps'
Fsh:
Ref This Cmd Output
self.py exps2 'select reqs,comment, netbytes from _all where app="search"'
Part two
reqs
comment
netbytes
head:100000-fixed:100@s2i2@reqs-SoGou
throughputreal1
4090.9054326
head:100000-fixed:100@s2i2@reqs-SoGou
throughputreal1
4090.9054326
Part three
head:100000-fixed:100@s2i2@reqs-SoGou
throughputreal1
6665.93762575
head:100000-fixed:100@s2i2@reqs-SoGou
throughputreal1
193224.978873
head:100000-fixed:100@s8i2-cycle@reqs-SoGou
throughputreal1
67895.3581801

Figure 3 snapshot of DCAngel's GUI

Figure.3 shows the snapshot of *DCAngel*'s GUI. The GUI can be divided into three parts. Part one is commands column. Each line in that column is a *DCAngel* command. Users can execute the command by **ctrl + left mouse button** click. Users can edit those commands to meet your requirement. Part two is command input column; you can input your command here and execute it by pressing **Enter**. Part three is a display column, which displays the result of the command.

Now we will show you the *DCAngel* command's grammar, so that you can writer your own commands.

A *DCAngel* command has two parts—a fixed part and a SQL like part. Let us look at the following command as an example.

self.py exps2 'select reqs, comment, netbytes from _all where app="search" '

The fixed part is self.py exps2 and the SQL like part is **'select reqs, comment, netbytes from _all where app="search" '**. For the SQL like part, users can write any statement that meets the sqlite3's syntax.

DCAngel's feedback may take a few seconds if it is your first time to execute a *DCAngel* command after a test. That is because *DCAngel* needs time to write metrics data it collected into database.

DCAngel also defines many extend SQL functions. Those functions usage are shown as below.

std(arg1) : standard deviation of arg1
corrcoef(arg1, arg2) : correlation coefficient between arg1 and arg2

correlate(arg1,arg2) : cross correlation of arg1 and arg2

wavg(arg1,arg2): weighted average of arg1, and arg2 is weight

xplot(arg1, arg2, arg3, arg4) : draw the scatter figure of arg4. The x-axis of this figure is time and the y-axis is arg4's average value. arg1 and arg2 should be "path" and "host" respective. arg3 is degree of data aggregation. If arg3 equals 100, each point in the figure represents the average value of 100 arg4.

xhist(arg1, arg2, arg3, arg4) : draw the histogram of arg4's occurrence times. The x-axis of this figure is occurrence times and the y-axis is arg4's average value. arg1 and arg2 should be "path" and "host" respective. arg3 is degree of data aggregation. If arg3 equals 100, each value on the x-axis represents the average value of 100 arg4.

xscatter(arg1,arg2,arg3,arg4,arg5) : draw bi-dimensional histogram of arg4 and arg5. arg1 and arg2 should be "path" and "host" respective. arg3 is degree of data aggregation. If arg3 equals 100, each value on x-axis and y-axis represents the average value of 100 arg4 and arg5.

xcorr(arg1,arg2,arg3,arg4,arg5) : plot the cross correlation between arg4 and arg5. arg1 and arg2 should be "path" and "host" respective. arg3 is degree of data aggregation.

If you want to use xplot you must make sure that the following read color words are not changed:

self.py exps2 'select reqs,comment,host, xplot(path, host, 1, $metric) from exps natural join all_events

self.py exps2 'select reqs,comment,host, xhist(path, host, 1, $metric) from exps natural join all_events

self.py exps2 'select reqs,comment,host, xscatter(path, host, 1, $metric,$metic) from exps natural join all_events

self.py exps2 'select reqs,comment,host, xcorr(path, host, 1, $metric,$metric) from exps natural join all_events

For $metric it can be any $metircs can be any field in Appendix B

We list the table structure of *DCAngel*'s database in Appendix A. Users can look up Appendix A and write your own *DCAngel* command.

4. Building your own DCAngel

If you want to build your own *DCAngel*, this part will give some advices. If following introductions do not suffice for your approach, you may contact us via jiazhen@ncic.ac.cn.

4.1. CPU

If your **Slave** nodes do not own a CPU whose type is one of the types we mentioned at section 3.2.6, you should modify line 167 to line 201 of *$DCAngel/hmon/hmon.py*.

```
167 kperf_events_map = '''
168 CPU_CLK_UNHALTED.CORE 3c                  # cpu_cycles
169 CPU_CLK_UNHALTED.BUS 13c                  # bus cycles
170 INST_RETIRED.ANY c0                       # insets
171 ITLB_MISS_RETIRED c9                      # itlb_misses
172 DTLB_MISSES.ANY 108                       # dtlb_misses
173 L1I_MISSES 81                            # icache_misses
174 L1D_REPL f45                             # dcache_misses
175 L2_LINES_IN.ANY f024                      # l2cache_misses
176
177 PAGE_WALKS.CYCLES    20c                 # page_walks
178 CYCLES_L1I_MEM_STALLED 86                 # icache_stalls
179
180 BR_INST_RETIRED.ANY c4                    # br_insts
181 BR_INST_RETIRED.MISPRED c5                # br_misses
182
183 INST_RETIRED.LOADS 1c0                    # load_insts
184 INST_RETIRED.STORES 2c0                   # store_insts
185 INST_RETIRED.OTHER 4c0                    # other_insts
186 SIMD_INST_RETIRED.ANY 1fc7               # simd_insts
187 FP_COMP_OPS_EXE 10                       # fp_insts
188
189 RESOURCE_STALLS.ANY 1fdc                  # res_stalls
190 RESOURCE_STALLS.ROB_FULL 1dc             # rob_stalls
191RESOURCE_STALLS.RS_FULL 2dc               # rs_stalls
192 RESOURCE_STALLS.LD_ST 4dc                # ldst_stalls
193 RESOURCE_STALLS.FPCW 8dc                 # fpcw_stalls
194 RESOURCE_STALLS.BR_MISS_CLEAR 10dc # br_miss_stalls
195
196 BUS_TRANS_ANY    e070                    # bus_trans
197 BUS_DRDY_CLOCKS 2062                     # bus_drdy
```

```
198 BUS_BNR_DRV 2061                    # bus_bnr
199 BUS_TRANS_BRD e065                   # bus_trans_brd
200 BUS_TRANS_RFO e066                   # bus_trans_rfo
201 '"
```

You should go over your CPU's software design manual and change hexadecimal number above to the corresponding CPU event number.

4.2. Separate DCAngel

You can use *DCAngel* only for collecting data or analysis data. The metrics data *DCAngel* collected could be found in *$DCAngel/exp/log* directory. Users can use another tool to analyze data.
Users can also use another tool to collect metrics data, and make sure that those data have the same format as *DCAngel*-collected data. Then users can visit DCAngel's browser port to get data and visualize data.

Appendix A –Metrics collected by DCAngel

variable	Definition
Metrics from performance counters	
cpu_cycles	Core cycles when core is not halted
bus_cycles	Bus cycles when core is not halted
insts	Retired instructions
itlb_misses	Retired instructions that missed the ITLB
dtlb_misses	Memory accesses that missed the DTLB
icache_misses	Instruction Fetch Unit misses
dcache_misses	L1 data cache misses
page_walks	Duration of page-walks in core cycles
icache_stalls	Cycles during which instruction fetches stalled
br_insts	Retired branch instructions
br_misses	Retired mispredicted branch instructions.
load_insts	Instructions retired, which contain a load
store_insts	Instructions retired, which contain a store
other_insts	Instructions retired, which no load or store operation
simd_insts	Retired Streaming SIMD instructions
fp_insts	Floating point computational micro-ops executed
res_stalls	Resource related stalls
rob_stalls	Cycles during which the reorder buffer full
rs_stalls	Cycles during which the reserve station full
ldst_stalls	Cycles during which the pipeline has exceeded load or store limit or waiting to commit all stores
fpcw_stalls	Cycles stalled due to floating-point unit control word writes
br_miss_stalls	Cycles stalled due to branch misprediction
bus_trans	All bus transactions
bus_drdy	Bus cycles when data is sent on the bus
bus_bnr	Number of Bus Not Ready signals asserted
bus_trans_brd	Burst read bus transactions
bus_trans_rfo	Read For Ownership bus transactions
Metrics from /proc filesystem	
usr	User mode CPU time
nice	The CPU time of processes whose nice value is negative
sys	Kernel mode CPU time
idle	Idle time
iowait	Iowait time
irq	Hard interrupt time
softirq	Soft interrupt time
intr	The times of interrupt happened
ctx	Context switch times
procs	Process number

running	The number of processes that is running
blocked	The number of processes that is blocked
mem_total	Total memory
free	Memory that is not used
buffers	Size memory in buffer cache
cached	Memory that cache used
swap_cached	Memory that once was swapped out, but still in the swapfile
active	Memory that has been used more recently
inactive	Memory that is not active
swap_total	Total amount of physical swap memory
swap_free	Total amount of free swap memory
pgin	The number of pages that paged in from disk
pgout	The number of pages that paged out to disk
pgfault	The number of page fault
pgmajfault	The number of major page faults
active_conn	TCP active connection
passive_conn	TCP passive connection
rbytes	Received bytes
rpackets	Received packets
rerrs	Received error packets number
rdrop	Number of packets dropped by native network adapter
sbytes	Bytes sent
spackets	Packets sent
serrs	Number of error packets sent
sdrop	Number of packets dropped by remote network adapter
read	Times of disk reads
read_merged	Times of disk merged reads
read_sectors	Times of sectors read
read_time	The total time disk read
write	Times of disk writes
write_merged	Times of merged disk writes
write_sectors	Times of sectors write
write_time	The total time of disk write

DCAngel collects those metrics per second and writes those values into exp/log/.

Appendix B – DCAngel database table structure

For the abbreviations' meaning in following table's, users can refer to Appendix A.

Table exps

field	Definition
path	The test performance data's path under exp/ directory
app	User used application's name
comment	The comment when user used to specify a
reqs	Request name
duration	The test's duration
host	Node's host name

Table _all

Field	Definition
path	The test performance data's path under exp/ directory
host	Node's host name
insts	The mean value of instruction number
cpi	Cycles per instruction
br_miss_ratio	Branch miss ratio
br_stall_ratio	Branch stall ratio
icache_stall_ratio	Icache stall ratio
tlb_stall_ratio	TLB stall ratio
dcaceh_stall_ratio	Dcache stall ratio
l2cache_stall_ratio	L2 Cache stall ratio
res_stall_ratio	Resource related stall ratio
rob_stall_ratio	Reorder buffer stall ratio
rs_stall_ratio	Reserve station stall ratio
ldst_stall_ratio	Load and store stall ratio
fpcw_stall_ratio	Float point unit stall ratio
br_mix	Branch instruction ratio
load_mix	Load instruction ratio
store_mix	Store instruction ratio
ldst_mix	Load and store instruction ratio
simd_mix	SIMD instruction ratio
fp_mix	Float point instruction ratio
other_mix	Instructions that except load and store ratio
bus_util	Bus utilization
bus_d_util	bus_drdy ratio （users can find bus_drdy and all the following abbreviations' meaning in Appendix A）
bus_bnr_ratio	bus_bnr ratio
bus_brd_ratio	bus_brd ratio
bus_rfo_ratio	bus_rfo_ratio

cpu_usage	CPU utilization
duration	The test's duration
netbytes	rnetbytes+snetbytes
netpackets	rnetpacket+snetpacket

The meaning of following field is the same as it in Appendix A. So we will not explain them here.
iowait
ctx
active
pgfault
pgmajfault
active_conn
passive_conn
read
write
read_sectors
write_sectors

For table _all, we also define some macros which you can use to simplify your inputting.
For example, you can write a *DCAngel* command self.py exps2 'select $prim from _all ', which has the same function with self.py exps2 'select app, comment, reqs, host from _all'

Macros and their definitions

macros	definition
$prim	app, comment, reqs, host
$hpc_basic	insts, cpi, br_miss_ratio
$stall_breakdown	br_stall_ratio, icache_stall_ratio, tlb_stall_ratio, dcache_stall_ratio, l2cache_stall_ratio, res_stall_ratio, rob_stall_ratio, rs_stall_ratio, ldst_stall_ratio, fpcw_stall_ratio
$inst_mix	br_mix, load_mix, store_mix, ldst_mix, simd_mix, fp_mix, other_mix
$cache	itlb_miss_ratio, dtlb_miss_ratio, icache_miss_ratio, dcache_miss_ratio, l2cache_miss_ratio
$bus	bus_util, bus_d_util, bus_bnr_ratio, bus_brd_ratio, bus_rfo_ratio
$proc_basic	cpu_usage, iowait, ctx, active, pgfault, pgmajfault
$net	active_conn, passive_conn, netbytes, netpackets,
$disk	read, write, read_sectors, write_sectors
$proc_selected	cpu_usage,iowait,ctx,active,pgmajfault,read_sectors
$hpc_all	$hpc_basic, $cache, $bus, $inst_mix
$proc_all	$proc_basic,$net,$disk